6.830/6.814 — Notes* for Lecture 1: Introduction to Database Systems

Carlo A. Curino

September 10, 2010

2 Introduction

READING MATERIAL: Ramakrishnan and Gehrke Chapter 1

What is a database? A database is a collection of structured data. A database captures an abstract representation of the domain of an application.

- Typically organized as "records" (traditionally, large numbers, on disk)
- and relationships between records

This class is about *database management systems (DBMS)*: systems for creating, manipulating, accessing a database.

A DBMS is a (usually complex) piece of software that sits in front of a collection of data, and mediates applications accesses to the data, guaranteeing many properties about the data and the accesses.

Why should you care? There are lots of applications that we don't offer classes on at MIT. Why are databases any different?

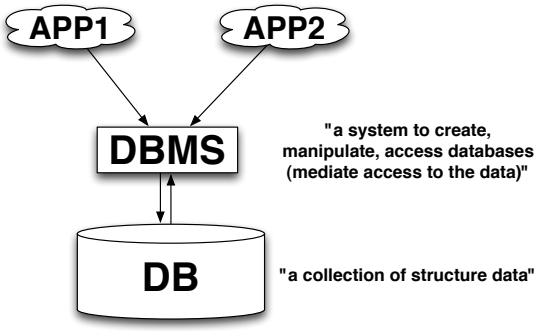

Figure 1: What is a database management system?

- Ubiquity (anywhere from your smartphone to Wikipedia)
- Real world impact: software market (roughly same size as OS market roughly \$20B/y). Web sites, big companies, scientific projects, all manage both day to day operations as well as business intelligence + data mining.
- *You need to know about databases if you want to be happy!*

The goal of a DBMS is to simplify the storing and accessing of data. To this purpose DBMSs provide facilities that serve the most common operations performed on data. The database community has devoted significant effort in formalizing few key concepts that most applications exploit to manipulate data. This provides a formal ground for us to discuss the application requirements on data storage and access, and compare ways for the DBMS to meet such requirements. This will provide you with powerful conceptual tools that go beyond the specific topics we tackle in this class, and are of general use for any application that needs to deal with data.

Now we proceed in showing an example, and show how hard is doing things without a DB, later we will introduce formal DB concepts and show how much easier things are using a DB.

3 Mafia Example

Today we cover the user perspective, trying to detail the many reason we want to use a DBMS rather than organizing and accessing data directly, for example as files.

Let us assume I am a *Mafia Boss* (Note: despite the accent this is not the case, but only hypothetical!) and I want to organize my group of "picciotti" (sicilian for the criminals/bad guys working for the boss, a.k.a the soldiers, see Figure 2) to achieve more efficiency in all our operations. I will also need a lot of book-keeping, security/privacy etc.. Note that my organization is very

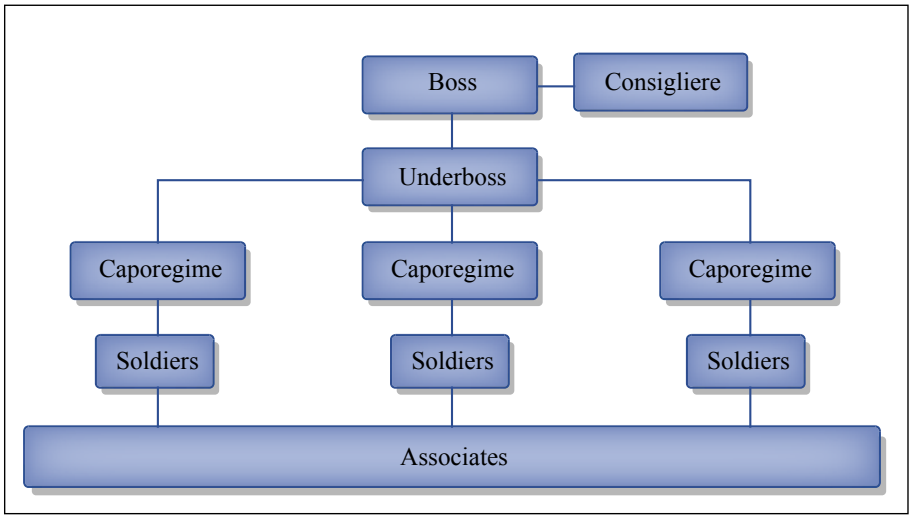

Image by MIT OpenCourseWare.

Figure 2: Mafia hierarchy.

large, so there is quite a bit of things going on at any moment (i.e., many people accessing the database to record or read information).

I need to store information about:

- people that work for me (soldiers, caporegime, etc..)
- organizations I do business with (police, 'Ndrangheta, politicians)
- completed and open operations:
  - protection rackets
  - arms trafficking
  - drug trafficking
  - loan sharking
  - control of contracting/politics
  - I need to avoid that any of may man is involved in burglary, mugging, kidnapping (too much police attention)
  - cover-up operations/businesses
  - money laundry and funds tracking
- assignment of soldiers to operations
- etc...

I will need to share some of this information with external organizations I work with, protecting some of the information.

Therefore I need:

- the boss, underboss and consigliere should be able to access all the data and do any kind of operations (assign soldiers to operations, create or shutdown operations, pay cops, check the total state of money movements, etc...)
- the accountants (20 of them) access to perform money book-keeping (track money laundering operations, move money from bank to bank, report bribing expenses)

- the soldiers (5000) need to report daily misdeeds in a daily-log, and report money expenses and collections
- the semi-public interface accessible by other bosses I collaborate with (search for cops on our books, check areas we already cover, etc..)

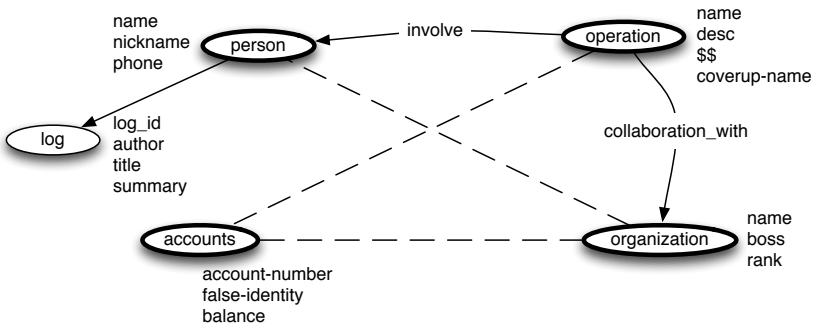

Figure 3: What data to store in my Mafia database.

3.1 An offer you cannot refuse

I make you an offer you cannot refuse: "you are hired to create my Mafia Information System, if you get it right you will have money, sexy cars, and a great life. If you get it wrong... well you don't want to get it wrong".

As a first attempt, you think about just using a file system:

1. **What to represent:**, what are the key entities in the real world I need to represent? how many details?
2. **How to store data:** maybe we can use just files: people.txt, organizations.txt, operations.txt, money.txt, daily-log.txt. Each files contains a textual representation of the information with one item per line.
3. **Control access credentials at low granularity:** accountants should know about money movement, but not the names and addresses of our soldiers. Soldiers should know about operations, but not access money information
4. **How to access data:** we could write a separate procedural program opening one or more files, scanning through them and reading/writing information in them.
5. **Access patterns and performance:** how to find shop we didn't collected money from for the longest time (and at least 1 month)? scan the huge operation file, sort by time, pick the oldest, measure time? (need to be timely or they will stop paying, and this get the boss mad... you surely

don't want that, and make sure no one is accessing it right now). "Tony Schifezza" is a mole, we need to find all the operations and people he was involved or knew about and shut them down... quick... like REAL quick!!!

6. **Atomicity**: when an accountant moves money from one place to another you need to guarantee that either money are removed from account A and added to account B, or nothing at all happens... (You do not want to have money vanishing, unless you plan to vanish too!).
7. **Consistency**: guarantee that the data are always in a valid state (e.g., there are no two operations with the same name)
8. **Isolation**: multiple soldiers need to add to daily-log.txt at the same time (risk is that they override each other work, and someone get "fired" because not productive!!)
9. **Durability**: in case of a computer crash we need to make sure we don't lose any data, nor that data get scrambled (e.g., If the system says the payment of a cop went through, we must guarantee that after reboot the operation will be present in the system and completed. The risk is police taking down our operation!)

Using the file system, you realize that most probably you will fail, and that can be very dangerous... Luckily you are enrolled in 6.830/6.814 and you just learned that: *Databases address all of these issues!!* you might have a chance! In fact, you might notice that the issues listed above are already related to the three concepts we mentioned before: 1-3 are problems related to *Data Model*, 4-5 are problems related to the *Query language* and 6-9 are problems related to *Transactions*.

So let's try to do the same with a "database" and get the boss what he needs.

3.2 More on fundamental concepts

Database are a microcosm of computer science, their study covers: languages, theory, operating systems, concurrent programming, user interfaces, optimization, algorithms, artificial intelligence, system design, parallel and distributed systems, statistical techniques, dynamic programming. Some of the key concepts we will investigate are:

Representing Data We need a consistent structured way to represent data, this is important for consistency, sharing, efficiency of access. From database theory we have the right concepts.

- *Data Model:* a set of constructs (or a paradigm) to describe the organization of data. For example tables (or more precisely *relations*), but we could also choose graph, hierarchies, objects, triples <subject,predicate,object>, etc..

- *Conceptual/Logical Schema:* is a description of a particular collection of data, using the a given data model (e.g., the schema of our Mafia database).
- *Physical Schema:* is the physical organization of the data (e.g., data and index files on disk for our Mafia database).

Declarative Querying and Query Processing a high-level (typically declarative) language to describe operations on data (e.g., queries, updates). The goal is to guarantee *Data independence (logical and physical)*, by separating "what" you want to do with data from "how" to achieve that (more later).

- High level language for accessing data
- "Data Independence" (logical and physical)
- Optimization Techniques for efficiently accessing data

Transactions

- a way to group actions that must happen atomically (all or nothing)
- guarantees to move the DB content from a consistent state to another
- isolate from parallel execution of other actions/transactions
- recoverable in case of failure (e.g., power goes out)

This provide the application with guarantees about a group of actions even in presence of concurrency and failures. It is a unit of access and manipulation of data. And significantly simplify the work of application developers.

This course covers these concepts, and goes deep into the investigation of how modern DBMS are designed to achieve all that. We will not cover the more artificial-inteligence / statistical / mining related areas that are also part of database research. Instead, we will explore some of the recent *advanced topics in database research*—see class schedule to get an idea of the topics.

3.3 Back to our Mafia database

What features of our organization shall we store? How do we want to capture them? Choose a level of abstraction and describe only the relevant details (e.g., I don't care about favorite movies for my soldiers, but I need to store their phone numbers). Let's focus on a subset:

- each person has real name, nickname, phone number
- each operation has a name, description, economical value, cover-up name
- info about the persons involved in an operation and their role,

We could represent this data according to many different data models:

- hierarchies
- objects
- graph
- triples
- etc..

Let's try using an XML hierarchical file:

```
<person>
    <name> </name>
    <nickname> </nickname>
    <phone> </phone>
    <operation>
            <op_name> </op_name>
            <description> </description>
            <econ_value> </econ_value>
            <coverup_name> </coverup_name>
    </operation>
</person>
```

Operations are duplicated in each person, this might make the update very tricky (inconsistencies) and the representation very verbose and redundant. Otherwise we can organize the other way around with people inside operations, well we would have people replicated.

Another possibility is using a graph structure with people, names, nicknames,phones, operation_names etc.. as nodes, and edges to represent relationships between them. Or we could have objects and methods on them, or triples like <carlo,is_a,person>, <carlo,phone,5554348882> etc..

Different data models are more suited for different problems.

They different expressive power and different strengths depending on what data you want to represent and how you need to access them.

Let's choose the relational data model and represent this problem using "tables". Again there are many ways to structure the representation, i.e., different "conceptual/logical schemas" that could capture the reality are modeling. For example we can have a single big table with all info together... again, is redundant and might slow down all the access to data.

The "database design" is the art of capturing a set of real world concepts and their relations in the best possible organization in a database. A good representation is shown in Figure 4. It is not redundant and contains all the information we care about.

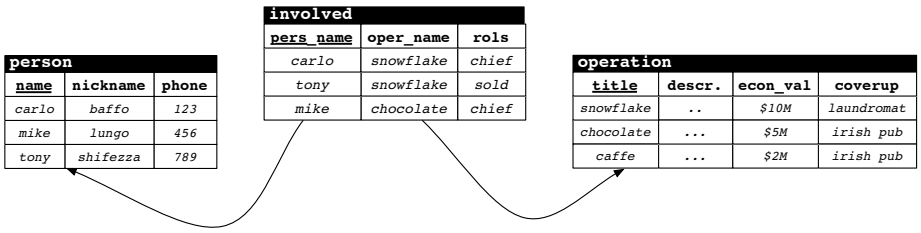

Figure 4: Simple Logical Schema for a portion of our Mafia database.

What about the physical organization of the data? As a database user you can ignore the problem, thanks to the physical independence! As a student of this class you will devote a lot of effort in learning how to best organize data physically to provide great performance to access data.

3.4 Accessing the data (transactionally)

As we introduced before databases provide high-level declarative query languages. The key idea is that you describe *"what"* you want to access, rather than *"how"* to access it.

Let's consider the following operations you want to do on data, and how we can represent them using the standard relational query language SQL:

- Which operations involve "Tony Schifezza"?

```
SELECT oper_name
FROM involved
WHERE person = "tony";
```

- Given the "laundromat" operation, get the phone numbers of all the people involved in operations using it as a cover up.

```
SELECT p.phone
FROM  person p, operation o, involve i
WHERE p.name = i.person AND
      i.oper_name = o.name AND
      o.coverup_name = "laundromat";
```

- Reassign Tony's operations to Sam and remove Tony from the database (he was the mole).

```
BEGIN
 UPDATE involved i SET pers_name="sam" WHERE pers_name="tony";
 DELETE FROM person WHERE name = "tony";
COMMIT
```

- Create a new operation with “Sam Astuto” in charge of it.

```
BEGIN
 INSERT INTO operation VALUES ('newop1','',0,'Sam's bakery');
 INSERT INTO involve VALUES ('newop1','sam','chief');
COMMIT
```

Let us reconsider the procedural approach. You might organize data into files: one record of each table in a file, and maybe sort the data by one of the fields. Now every different access to the data, i.e., every “query” should become a different program opening the files, scanning them, reading or writing certain fields, saving the files.

4 Extras

The two following concepts have been broadly mentioned but not discussed in details in class.

Optimization The goal of a DBMS is to provide a library of sophisticated techniques and strategy to store, access, update data that also guarantees performance, atomicity, consistency, isolation, durability. DBMS automatically compile the user declarative queries into an *execution plan* (i.e., a strategy that applies various steps to achieve the compute the user queries), looks for equivalent but more efficient ways to obtain the same result *query optimization*, and execute it, see example in Figure 5.

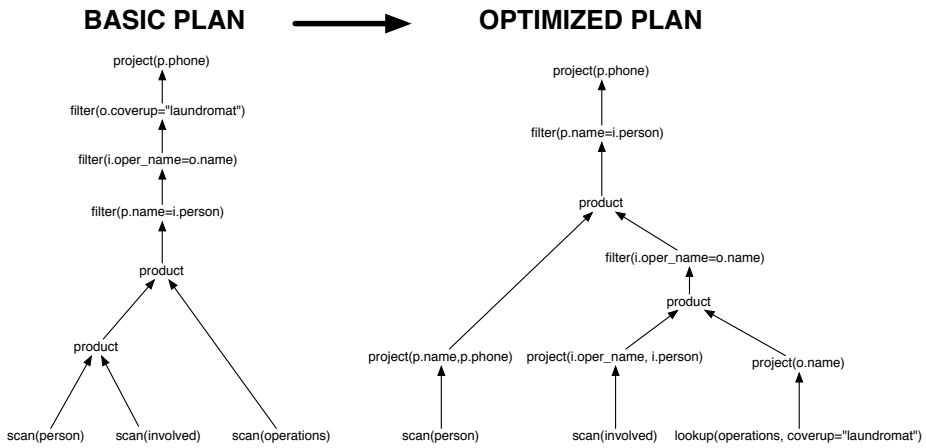

Figure 5: Two equivalent execution plan, a basic and an optimized one.

External schema A set of views over the logical schema, that predicates how users see/access data. (e.g., a set of views for the accountants). It is often not physically materialized, but maintain as a view/query on top of the data.

Let try to show only coverup names of operations worth less or equal to $5M and the nicknames of all people involved using a view (see Figure 6):

```
CREATE VIEW nick-cover AS
SELECT  nickname, coverup_name
FROM operation o, involved i, person p
WHERE p.name = i.person AND
      i.oper_name = o.name AND
      o.econ_val <= 5M;
```

nick-cover

nickname	coverup
baffo	*laundromat*
lungo	*irish pub*
schifezza	*laundromat*

Figure 6: Simple External Schema for a portion of our Mafia database.

5 What's next?

Next week lessons introduce more formally the relational model (and some of its history) and how to design the schema of a database. After that we will dive into the DBMS internals and study "how" DBMS are internally architected to achieve all the functionalities we discussed. Later on we will study how to guarantee transactional behaviors, and how to scale a DBMS beyond a single node. The last portion of the course is devoted to more esoteric topics from recent advances in database research.

MIT OpenCourseWare
http://ocw.mit.edu

6.830 / 6.814 Database Systems

Fall 2010

For information about citing these materials or our Terms of Use, visit: http://ocw.mit.edu/terms.